

Winter 2014

A Method for Clustering High-Dimensional Data Using 1D Random Projections

Sangchun Han
Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/open_access_dissertations



Part of the [Computer Engineering Commons](#)

Recommended Citation

Han, Sangchun, "A Method for Clustering High-Dimensional Data Using 1D Random Projections" (2014). *Open Access Dissertations*. 280.
https://docs.lib.purdue.edu/open_access_dissertations/280

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Sangchun Han

Entitled

A Method for Clustering High-Dimensional Data Using 1D Random Projections

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

MIREILLE BOUTIN

Chair

DAVID J. LOVE

EDWARD J. DELP

XIAOJUN LIN

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): MIREILLE BOUTIN

Approved by: V. Balakrishnan 11-11-2014
Head of the Graduate Program Date

A METHOD FOR CLUSTERING HIGH-DIMENSIONAL DATA USING 1D
RANDOM PROJECTIONS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Sangchun Han

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2014

Purdue University

West Lafayette, Indiana

TABLE OF CONTENTS

	Page
LIST OF TABLES	iii
LIST OF FIGURES	iv
ABSTRACT	v
1 INTRODUCTION	1
2 PROPOSED CLUSTERING METHOD	8
2.1 Binary Clustering	8
2.2 Rationale for Proposed Binary Clustering	10
2.3 Hierarchical Clustering	14
3 FAST 1D 2-MEANS CLUSTERING	18
3.1 Step 1: Sorting-Based Algorithm	19
3.2 Step 2: Bucket-Based Algorithm	22
3.3 Step 3: Partial Quicksort to Construct Buckets	23
4 EXPERIMENTS	25
4.1 Efficiency	27
4.2 Effectiveness	30
4.3 Very High-Dimensional Data	39
5 CONCLUSION	42
REFERENCES	44
A DERIVATIONS	47
VITA	49

LIST OF TABLES

Table	Page
2.1 Data description.	11
4.1 Description of high-dimensional datasets used in experiments.	26
4.2 Description of very high-dimensional datasets used in experiments. . .	26
4.3 RP1D parameters.	27
4.4 Run-time comparison using PENDIGITS dataset.	29
4.5 Experimental result using GLASS dataset.	31
4.6 Experimental result using VOWEL dataset.	32
4.7 Experimental result using PENDIGITS dataset.	33
4.8 Experimental result using SHAPE dataset.	34
4.9 Experimental result using DIABETES dataset.	35
4.10 Experimental result using LIVER dataset.	36
4.11 Experimental result using BREAST dataset.	37
4.12 Comparison with existing subspace clustering methods.	38
4.13 Comparison using LIBRAS dataset.	40
4.14 Comparison using MFEAT-FAC dataset.	41

LIST OF FIGURES

Figure	Page
1.1 Two equally weighted Gaussian random vectors in 2D and its distance distribution (bimodal distribution).	3
1.2 One Gaussian random vector in 2D and its distance distribution (unimodal distribution).	3
1.3 Test case illustrating the difficulty of clustering high-dimensional data.	5
1.4 Test case illustrating the similarity between increasing space dimension and decreasing cluster distance.	6
2.1 Data matrix $D_{n \times n}$	8
2.2 Binary clustering using a 1D random projection.	9
2.3 Repetition of 1D random projection.	10
2.4 Experimental results using MUSK dataset.	11
2.5 Experimental results using LIBRAS dataset.	12
2.6 Experimental results using MFEAT-FAC dataset.	12
2.7 Experimental results using ARRHYTHMIA dataset.	12
2.8 Tree structure obtained by applying the binary clustering method recursively.	15
2.9 Flowchart of the top-down hierarchical clustering using 1D random projections.	16
3.1 Complexity of 1D 2-means clustering.	20
3.2 Illustration of the bucket-based 1D 2-means clustering algorithm. . . .	23
4.1 Runtime comparison between k-means clustering and fast 1D 2-means clustering.	29

ABSTRACT

Han, Sangchun PhD, Purdue University, December 2014. A Method for Clustering High-Dimensional Data Using 1D Random Projections. Major Professor: Mireille Boutin.

Clustering high-dimensional data is more difficult than clustering low-dimensional data. The problem is twofold. First, there is an efficiency problem related to the data size, which increases with the dimensionality. Second, there is an effectiveness problem related to the fact that the mere existence of clusters in sample sets of high dimensions is questionable, as empirical samples hardly tend to cluster together in a meaningful fashion. The current approach to addressing this issue is to seek clusters in embedded subspaces of the original space. However, as dimensionality increases, a naive exhaustive search among all subspaces becomes exponentially more complex, which leads to an overwhelming time complexity. We propose an alternative approach for high-dimensional data clustering. Our solution is a top-down hierarchical clustering method using a binary tree of 1D random projections. As real data tends to have a lot of structures, we show that a 1D random projection of real data captures some of that structure with a high probability. More specifically, the structure manifests itself as a clear binary clustering in the projected data (1D). Our approach is efficient because most of the computations are performed in 1D. To increase efficiency of our method even further, we propose a fast 1D 2-means clustering method, which takes advantage of the 1D space. Our method achieves a better quality of clustering as well as a lower run-time compared to existing high-dimensional clustering methods.

1. INTRODUCTION

Clustering is the problem of finding natural groupings in a set of data points. There are many methods for clustering data points, e.g., k-means [1, 2], kernel k-means [3], expectation-maximization algorithm [4], BIRCH [5] and DBSCAN [6]. However, these methods do not work well in high-dimensional space.

There are two problems in high-dimensional data clustering: efficiency and effectiveness. The efficiency problem comes from the fact that the data size linearly increases as the dimensionality increases. The effectiveness problem is more subtle, as we now explain.

A high-dimensional space is different from a low-dimensional space in many ways. For example, as the dimensionality increases, the distance to the nearest neighbor and the distance to the farthest neighbor becomes close to each other [7]. Thus, distance-based clustering algorithms often do not work in high-dimensional spaces. One popular method of distance-based clustering is k-means, in which each data point is assigned to the nearest among k centroids at each iteration. However, in high-dimensional space, the distances from a data point to every centroid become nearly equal. So the clusters found by k-means may not correspond to meaningful structure in the data.

Another characteristic of high-dimensionality, called the curse of dimensionality, is the fact that even large datasets are sparse in high-dimensional spaces. This is an issue for clustering methods based on density estimation. When we count the number of points in hypercubes, we typically find that most of them are empty in high-dimensional space. This explains why density-based clustering methods like DBSCAN are likely to fail to cluster properly in high-dimensional space. More specifically, clusters obtained with a density-based clustering method are likely overfitted due to the sparse nature of the data points in high-dimensional space.

The use of distance-based and density-based clustering methods is problematic because they will often cluster the data even if there is no meaningful cluster in the data. Therefore, one may find clusters in high-dimensional space, but they may not be real or meaningful. In dimension two or three, one can simply visualize the result to check if it is correct. In high dimension, this is not possible. Instead, one uses a mathematical measure called “clustering tendency” or “clusterability” to determine, prior to clustering, whether a dataset contains clusters.

There are three widely used methods for measuring clustering tendency: a “distance distribution” method, a “spatial histogram” method, and a method called “Hopkins statistic” [8]. To understand how these methods work, let us first focus on the distance distribution method. In that method, the distance between a pair of points drawn at random, independently, from the dataset is considered. After drawing a large number of pairs, a histogram of distances is drawn and compared with that of a set of reference points drawn from a uniform distribution, which works as a “null model”. If the dataset contains two clusters for example, then the distribution of distances typically features two separated “bumps.” In contrast, if the data points are drawn from a single cluster, as is the case for the reference points, then the distribution of distances typically forms a single “blob.”

These are illustrated in Figure 1.1 and Figure 1.2. Figure 1.1 shows 500 samples drawn independently from one of two Gaussians in 2D with equal probability. The distance between the means of the Gaussians is 6 and the standard deviation matrix for each Gaussian is the identity matrix. The second graph of Figure 1.1 shows the distribution of the distance between a pair of points drawn independently from this Gaussian mixture. Notice the two bumps (bimodal distribution). In contrast, if 500 samples are drawn following an single Gaussian in 2D (see the first graph of Figure 1.2), then the pairwise distance between two independent samples has a unimodal distribution, as shown in the second graph of Figure 1.2.

This is the idea behind using the distance distribution as a clustering tendency. Just by observing the distance distribution, it seems to be possible to know whether

there are clusters in the data or not. However, this is only when the distance between clusters is large enough in comparison of the dimensionality because the intra-cluster distances and the inter-cluster distances are intermingled as dimensionality increases. To illustrate this problem, we performed the following experiments.

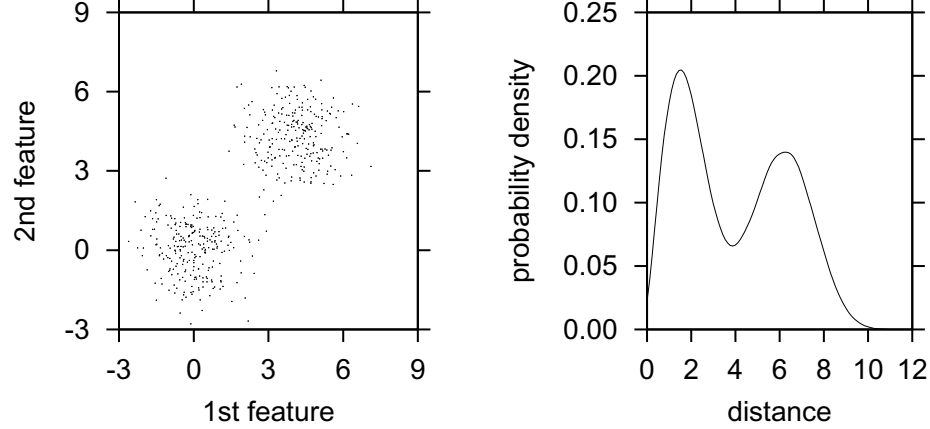


Fig. 1.1.: Two equally weighted Gaussian random vectors in 2D and its distance distribution (bimodal distribution). The distance between the two centers is 6 and both standard deviation matrices are equal to the identity matrix.

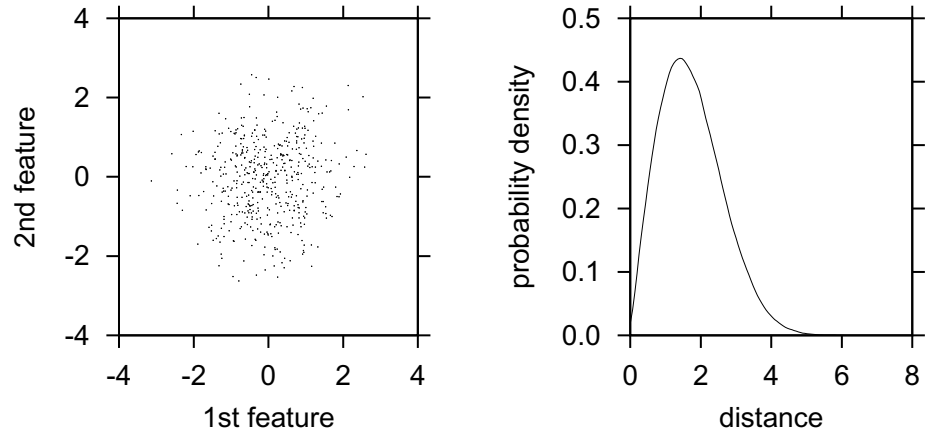


Fig. 1.2.: One Gaussian random vector in 2D and its distance distribution (unimodal distribution). The standard deviation matrix is the identity matrix.

We considered a mixture of two Gaussians in \mathbb{R}^m consisting of one Gaussian centered at the origin and the other Gaussian centered at $\mu = \frac{6}{\sqrt{m}}(1, 1, \dots, 1)$. Thus the distance between two Gaussians was fixed to 6 regardless of the dimensionality m of the space. An equal weight was assigned to each Gaussian and their covariance matrix was set to the identity matrix. We then drew 500 pairs of points, independently following this Gaussian mixture model probability law, and computed the Euclidean distance between each pair of points. An approximation of the probability density function corresponding to these 500 sample distances is drawn in Figure 1.3 for three different choices of dimension ($m = 2, 15$, and 40). For small values of m (e.g., $m = 2$), the intra-cluster distances tends to be smaller than the inter-cluster distances, as indicated by the presence of the bimodal distribution in the probability density function of the distance between a pair of points. As m increases, the probability density function eventually becomes unimodal, as the distinction between intra-cluster distances and inter-cluster distances disappears. Note that this phenomenon is not specific to the Euclidean distance: it is related to properties of the topology of high-dimensional spaces, regardless of how this topology is defined.

As statistical pattern recognition teaches us, in a space of fixed dimension, the amount of separation between two Gaussians of equal weight with identical covariance matrices is determined by the distance between their means: as the distance between the means decreases, points drawn from the two Gaussians tend to be more and more intermingled and difficult to separate. The merging of clusters as the dimensionality increases is analogous to the merging of clusters as their distance decreases. To illustrate this, let us consider a mixture of two Gaussians with equal weight in a space of fixed dimension $m = 2$. We fix the covariance matrix for each Gaussian to the identity matrix and put the center of the first Gaussian at the origin. Then we place the second Gaussian at a variable distance d from the first by fixing its mean at $\mu = \frac{d}{\sqrt{2}}(1, 1, \dots, 1)$. As the distance d decreases, we expect the intra-cluster distances to become more and more similar to the inter-cluster distances. Indeed, when $d = 6$, the probability distribution function of the distance between a pair of points drawn

independently following this Gaussian mixture is clearly bimodal. But as the distance between clusters decreases, that probability distribution function becomes unimodal (e.g., when $d = 2$), as illustrated in Figure 1.4. Thus, we can conclude that high dimensionality has similar effect as small inter-cluster distance.

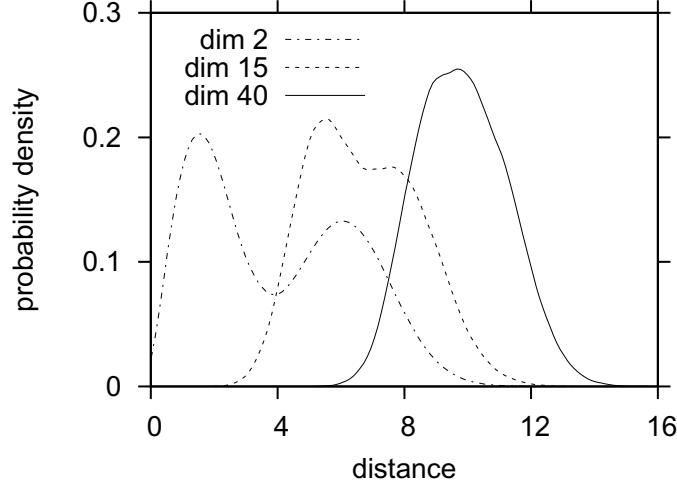


Fig. 1.3.: Test case illustrating the difficulty of clustering high-dimensional data. Two Gaussians at a fixed distance, $d = 6$, are embedded in a space of increasing dimension ($m = 2, 15$, and 40). As the dimension increases, the distinction between intra-cluster distance and inter-cluster distance disappears, and the pdf of the distance between two independent samples goes from a bimodal distribution to a unimodal distribution.

The problem of clustering tendency is not limited to the distance distribution; the Hopkins statistic method has the same problem. Similarly, the spatial histogram method may not work in high-dimensional space because, as we mentioned previously, most hypercubes in spatial histogram have no data points in high dimension. Thus, measuring the clustering tendency is often not an effective way to determine the existence of clusters in a high-dimensional space.

The effectiveness problem can be solved using the idea of embedded structures (subspace clusters). In other words, one can look for clusters embedded in low-dimensional subspaces of the given high-dimensional space. To illustrate this, let us

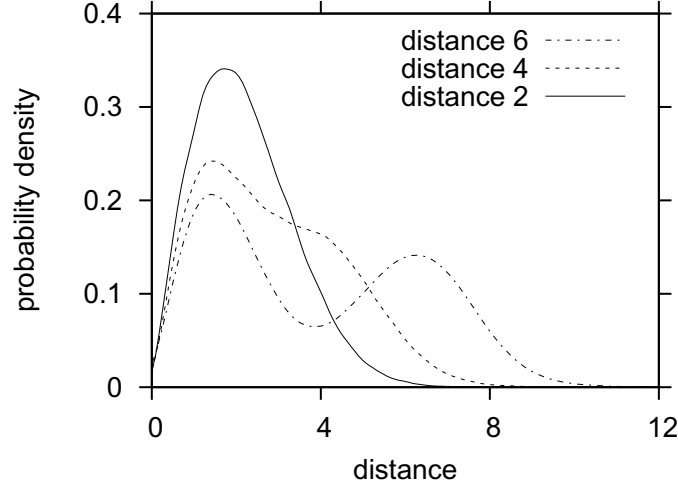


Fig. 1.4.: Test case illustrating the similarity between increasing space dimension and decreasing cluster distance. In a space of fixed dimension ($m = 2$), when two Gaussians are getting closer and closer together, the pdf of the distance between two independent samples goes from a bimodal distribution to a unimodal distribution. Thus, the distinction between intra-cluster distance and inter-cluster distance disappears in a similar fashion as when the dimension of the space increases.

think about three data points in \mathbb{R}^4 : $(1, 1, 2, 2)$, $(4, 1, 2, 4)$, and $(0, 2, 4, 1)$. While these do not form any particular structure in \mathbb{R}^4 , when restricted to the second and third dimensions, the first two data points then merge together. This embedded structure is not visible in the original space because the distance between the first and the third data points is $\sqrt{7}$, which is smaller than the distance between the first and the second data points, 5, in the original space.

One can view the phenomenon as the overwhelming noise (the first and the fourth dimensions) for the signal (the second and the third dimensions). The reason that the subspace cluster is not visible is that the noise is stronger than the signal in the original space.

As we just saw, this problem occurs in low-dimensional space, too. However, it is more problematic in high-dimensional space because it is more likely that more attributes (dimensions) work as noise to prohibit revealing the embedded structure.

Thus, one way to find clusters in data is to restrict the data to some dimensions and to look for clusters in those dimensions. In low-dimensional space, this can be done by performing an exhaustive search over all possible subdimensions. However, in dimension m , there are $2^m - 1$ possibilities of subdimensions. Thus, searching space increases exponentially as dimensionality increases. In other words, by trying to solve the effectiveness problem, we made the efficiency problem worse.

In the following, we propose an alternative approach in which we project the data onto a random vector thus reducing the data to one-dimension. We then find binary clusters in the projected (one-dimensional) data. This binary clustering method can capture partial embedded structures of the data. We show that it actually works well with real datasets. To extend it for any number of clusters, we use this binary clustering method as a component in a top-down hierarchical clustering. Our method is naturally efficient because most of the operations are performed in 1D. To further increase the efficiency, we propose a fast 1D 2-means clustering. In Chapter 4, we compare our method with 10 other subspace clustering methods [9] using 9 datasets. Our experimental results indicate that our method is, overall, the fastest and the most accurate.

Note that the concept of random projection [10,11] has been popularized in another context, namely dimension reduction. The objective in those context is to preserve the structure of the high-dimensional space using Johnson-Lindenstrauss lemma [12]. We use random projection differently. First, we do not try to preserve the overall information in the high-dimensional space, but rather we try to reveal the embedded structure that might be hidden in the high-dimensional space. In addition, we only use 1D random projection.

2. PROPOSED CLUSTERING METHOD

To simplify the discussion, we put the data points to cluster into a matrix called the data matrix. We denote the data matrix by $D_{n \times m}$ (Figure 2.1) where n is the number of rows and m is the number of columns. The row vectors are for data points (samples) while the column vectors are for features (attributes or dimensions).

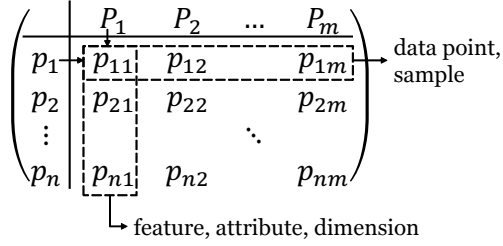


Fig. 2.1.: Data matrix $D_{n \times m}$.

The output of clustering is a partition of the indices $(1, 2, \dots, n)$ or a list of groups, where each group consists of indices. Note that the groups need not be disjoint.

2.1 Binary Clustering

We are given n data points, $p_1, \dots, p_n \in \mathbb{R}^m$, with m potentially very large. In this section, we describe a fast method for dividing the points into two disjoint clusters. As illustrated in Figure 2.2, we do this by partitioning the set of indices $C = (1, 2, \dots, n)$ into two disjoint subsets, C_1 and C_2 such that $C_1 \cup C_2 = C$.

To partition C , we generate a random (column) vector $v \in \mathbb{R}^m$ whose coordinates are drawn independently from a uniform distribution in the interval $[-1, 1]$. Denote by x_i the projection of p_i into v :

$$x_i = p_i \cdot v.$$

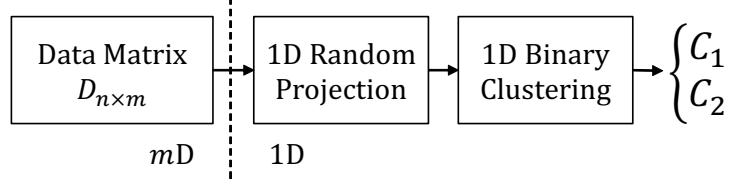


Fig. 2.2.: Binary clustering using a 1D random projection.

After stacking the data points p_1, \dots, p_n into a data matrix, $D_{n \times m}$, the projections x_1, \dots, x_n can be obtained by matrix multiplication:

$$X = D_{n \times m} \cdot v.$$

We then cluster the projected points x_i . Since the x_i are in \mathbb{R}^1 , this is a 1D clustering problem. The advantage of working in 1D is that the data has a natural ordering, which can be exploited to cluster faster. In essence, binary clustering in 1D corresponds to finding a threshold. In section 3, we propose a fast method for doing so. The method is based on minimizing the sum of two within-cluster sums of squares (“*withinss*”) [13]:

$$R = \{C_1, C_2\} = \arg \min_{C_1, C_2} \sum_{i \in C_1} (x_i - \mu_1)^2 + \sum_{i \in C_2} (x_i - \mu_2)^2, \quad (2.1)$$

where μ_1 and μ_2 are the means of C_1 and C_2 , respectively.

If the data obtained after projection is not well separated, then we can repeat the clustering process with a newly generated random vector v . For simplicity, we fix a predetermined number of repetition r , and pick the best clustering among those r . This process is summarized in Figure 2.3.

We measure the quality of a clustering using a rescaled version of *withinss* which we call “*normalized withinss*”,

$$W(x_1, \dots, x_n) = \min_{C_1, C_2} \frac{\sum_{i \in C_1} (x_i - \mu_1)^2 + \sum_{i \in C_2} (x_i - \mu_2)^2}{\sigma^2 \cdot n}, \quad (2.2)$$

where σ is standard deviation of x_1, \dots, x_n . Observe that W is small only when the data can be well separated into two clusters. Thus, we choose the clustering for which

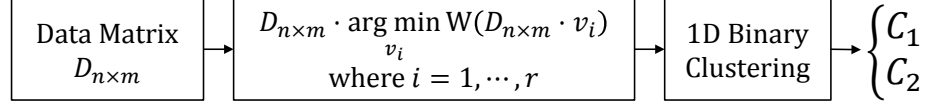


Fig. 2.3.: Repetition of 1D random projection to increase the chance of finding real clusters. r is the number of repetition of the 1D random projection. $v_i \in \mathbb{R}^m$ is the i_{th} random vector. Note that v_i is a column vector. So the second component produces a column vector with n elements. We use 1D 2-means clustering (Chapter 3) for 1D binary clustering.

W is minimum among all r clusterings. This strategy increases the chance to find real clusters even if each random projection has low success rate. For example, if the success rate of 1D random projection is 0.5%, its overall success rate is over 39% after 100 runs and over 99% after 1000 runs.

2.2 Rationale for Proposed Binary Clustering

Our proposed binary clustering method is based on the empirical observation that real data seems to have a very high tendency to cluster after a random projection. This seems to indicate that real data tends to have a lot of structure. We hope that the following examples clearly illustrate this.

The proposed binary clustering using 1D random projection works only when 1D random projection produces 1D distribution which can be partitioned into two clusters. If the dataset has no structure or very simple structure that we discussed in the introduction, the performance of 1D random projection is limited. On the other hand, if the dataset embeds complex structure, there are several possible 1D projections to reveal the structure and 1D random projection has more chance to find those.

We categorize datasets into three. The first group has a bimodal distance distribution. The second and third groups have a unimodal distance distribution. The

Table 2.1.: Data description.

Category	Name	Size	Dim	Size \times Dim
Category 1	MUSK	6598	166	1095268
	USAGE	10103	71	717313
	HUMAN	7352	561	4124472
Category 2	LIBRAS	360	90	32400
	MFEAT-FAC	2000	216	432000
	COREL	62480	64	3998720
Category 3	ARRHYTHMIA	452	279	126108
	SPAMBASE	4601	57	262257
	STOCK	3877	60	232620

difference between the second and the third groups is on the distribution of *normalized withinss* and the result of 1D random projection. All datasets except Stock are from [14]. Stock dataset [15]¹ contains sixty days of daily returns from Oct. 28, 2008 to Jan. 26, 2009. We remove 10% of stocks with low market capital.

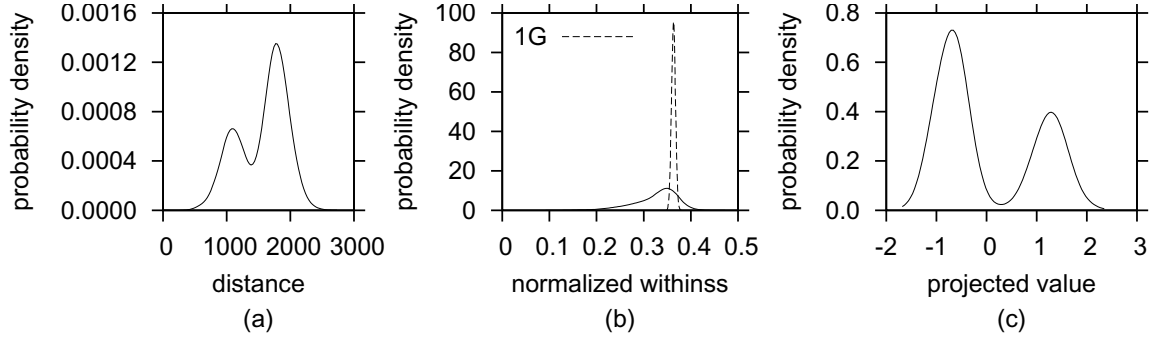


Fig. 2.4.: Experimental results using MUSK dataset.

¹The resulting dataset is available at <https://engineering.purdue.edu/~mboutin/RP1D>

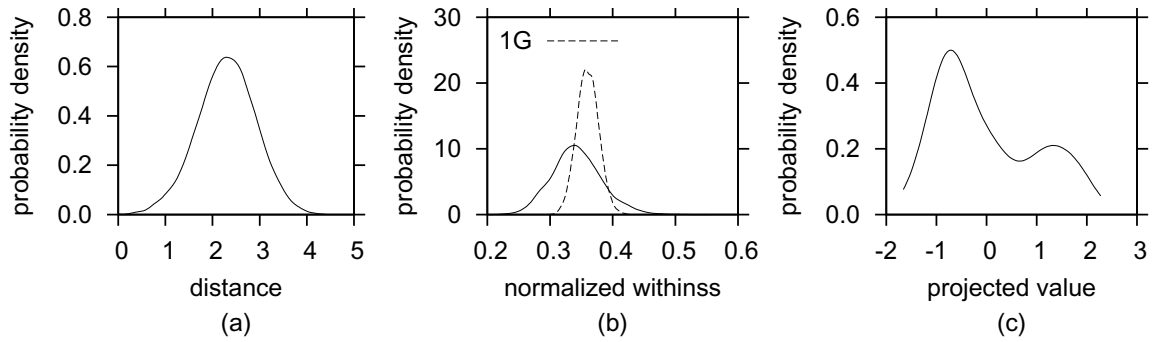


Fig. 2.5.: Experimental results using LIBRAS dataset.

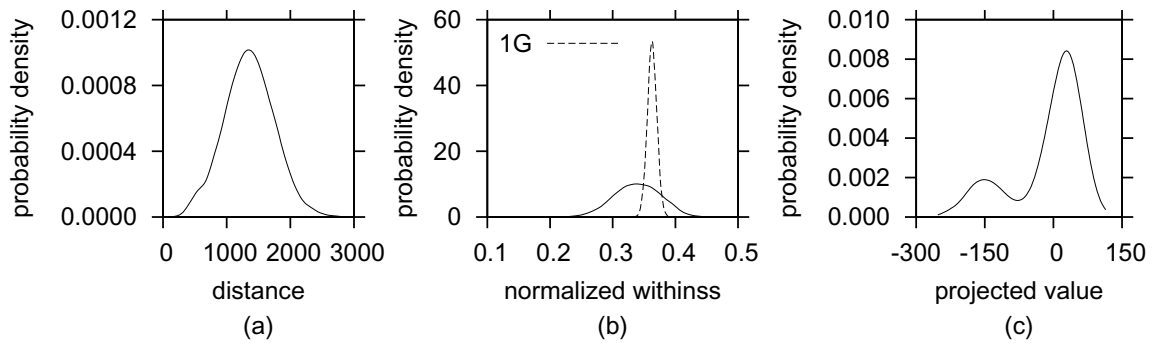


Fig. 2.6.: Experimental results using MFEAT-FAC dataset.

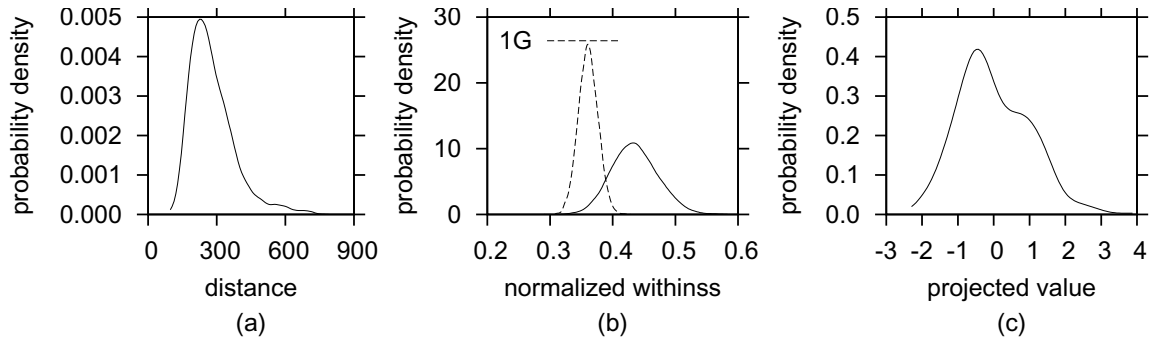


Fig. 2.7.: Experimental results using ARRHYTHMIA dataset.

We choose one dataset from the first category (MUSK), one from the third category (ARRHYTHMIA), and two from the second category (LIBRAS and MFEAT-FAC). The results of our experiments are illustrated in Figure 2.4 – Figure 2.7.

The first graph of each figure is the distance distribution. The second graph shows the distribution of the *normalized withinss* values obtained after having drawn 10,000 random vectors. A dotted line is plotted on top of our results. That line represents the pdf of *normalized withinss* from a reference data matrix containing the same number of data points as the dataset considered, where the data points are drawn from a standard normal distribution in a space with the same dimension as the dataset. The third graph shows the projection with the smallest *normalized withinss* value among all 10,000 projections.

Notice that MUSK has a bimodal distance distribution while other three datasets have a unimodal distribution. In the bimodal distance distribution, the left hump corresponds to intra distances within clusters and the right hump to inter distances between clusters. For the first category datasets, it is likely that traditional clustering algorithms would work to partition the dataset into two or more clusters. However, our clustering algorithm is still useful for this type of datasets as the third graph shows a clear separation between two clusters. In other words, regardless of the number of real clusters, it finds two groups of clusters, which are very different each other. There might be many possible combination in the grouping of clusters into two clusters. Our algorithm tries to find the best one in terms of *normalized withinss*. We conjecture that if there are only two real clusters, our 1D random projection method reflects those two real clusters.

The second graph for the MUSK dataset (Figure 2.4) shows that the MUSK *normalized withinss* distribution is very different from the result of the reference data matrix. Note that the average for the reference data matrix is always approximately 0.36. In some runs, *normalized withinss* is significantly smaller than 0.36. This suggests that some 1D random projections yield clear separation between two clusters. In addition, the values of *normalized withinss* are very dispersed. In other words, each 1D random projection produces very different results in terms of *normalized withinss*. This shows not only the effectiveness of 1D random projection but also the complex structure of MUSK dataset. Indeed, if the structure was simple, the 1D random projections would

not yield such a dispersed *normalized withinss* distribution, as we discussed in the introduction.

We see in Figure 2.5 and Figure 2.6 that LIBRAS and MFEAT-FAC have a unimodal distance distribution. If we used this as a measure of clustering tendency, we would conclude that there is no cluster. However, when we draw the *normalized withinss* distribution, we see clearly that this dataset embeds some structure that many 1D random projections can reveal. The best projected value distribution confirms that 1D random projection can catch some structure to partition into two clusters (Figure 2.5 (c) and Figure 2.6 (c)).

We see in Figure 2.7 that the ARRHYTHMIA dataset has also a unimodal distance distribution, but one with a tail. This might come from data points that are placed very far from most of other data points. When we look at the *normalized withinss* distribution (Figure 2.7 (b)), we see that it is similar to LIBRAS but shifted to the right. Indeed, most of the time, *normalized withinss* was larger than 0.36. Evidently, ARRHYTHMIA does have some kind of structure (otherwise its *normalized withinss* distribution would correspond more or less to that of the reference data matrix) although it is hard to describe what kind. However, by picking the projection for which *normalized withinss* is the smallest, we do obtain a value smaller than 0.36 and so we do find an acceptable separation (Figure 2.7 (c)).

2.3 Hierarchical Clustering

We proposed the binary clustering method using 1D random projections and showed that real datasets have complex structure that 1D random projections can exploit in the previous two sections. But this method is limited because it can find exactly only two clusters. To extend this for any number of clusters, there are two things we need to discuss further.

The first thing is applying the binary clustering method recursively. This yields a tree structure as shown in Figure 2.8. In other words, we obtain a top-down hierarchical clustering. Each leaf will be regarded as a cluster.

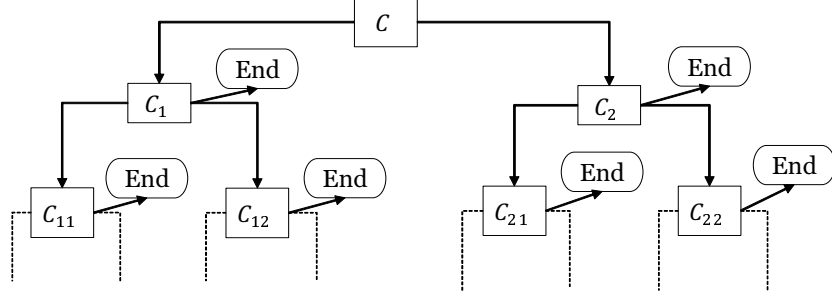


Fig. 2.8.: Tree structure obtained by applying the binary clustering method recursively.

The second thing is the termination condition of this recursive process, a required component of any hierarchical clustering. (Note that clustering tendency measures can be used for that purpose.) We propose two criteria for this purpose. Both of them require a reference data matrix filled with standard normal random variates. Using a reference data matrix, we have two sets of *normalized withinss*: $\{w_i\}_r$ and $\{w'_i\}_r$.

The first criterion uses histograms. From $\{w_i\}_r$ and $\{w'_i\}_r$, we construct two histograms and compare them to see how much they are different from each other. If the difference is large, we can say that the set of data points still embeds complex enough structure for further clustering. We find the maximum difference between two corresponding bins in two histograms divided by the number of data points and call it **ldiff**.

The second criterion is focusing of minimum *normalized withinss*. If it is significantly smaller than the average of the reference distribution (taking standard deviation into account), we assume that the set of data points has complex enough structure in it. More specifically, we set a threshold as the quantity:

$$\frac{\min(\{w_i\}_r) - \text{avg}(\{w'_i\})}{\text{std}(\{w'_i\}_r)},$$

where $\min(\cdot)$ finds the minimum value, $\text{avg}(\cdot)$ calculates the average, and $\text{std}(\cdot)$ calculates the standard deviation.

Using the binary tree structure and the termination condition, we have the top-down hierarchical clustering algorithm described in Algorithm 1. Figure 2.9 shows the flowchart of the hierarchical clustering using the binary clustering method. It describes the recursive nature of the algorithm while Algorithm 1 implements the algorithm with iterative routine for efficiency. This figure works as a component of each node in Figure 2.8.

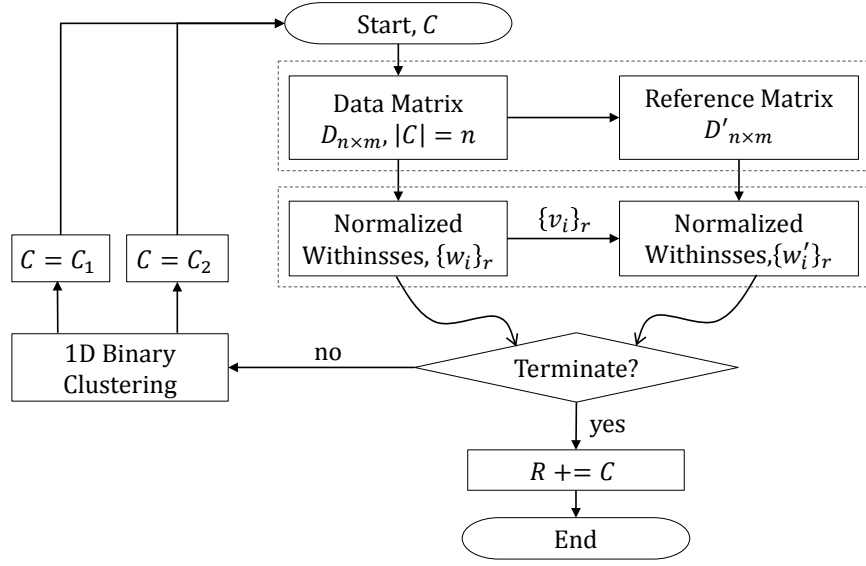


Fig. 2.9.: Flowchart of the top-down hierarchical clustering using 1D random projections.

Algorithm 1 Top-down hierarchical clustering using 1D random projections. currLayer and nextLayer are stack data structure, which has push and pop methods. RP1DBinary denotes a binary clustering using 1D random projection. Its input parameter is a data matrix, the indices of interest, and the number of random projections. It yields two result: **ldiff** and clustering result.

Require: $D_{n \times m}$: data matrix,
 t : threshold,
 s : the smallest possible cluster size,
 r : the number of repetition

- 1: currLayer.push($\{1, \dots, n\}$)
- 2: **repeat**
- 3: goFurther \leftarrow false
- 4: **for** com \leftarrow currLayer.pop() **do**
- 5: clusterer \leftarrow
 new RP1DBinary($D_{n \times m}$, com, r)
- 6: **if** clusterer.ldiff() $< t$ || com.size() $< s$ **then**
- 7: result.push(com)
- 8: **else**
- 9: goFurther \leftarrow true
- 10: nextLayer.pushAll(clusterer.result())
- 11: **end if**
- 12: **end for**
- 13: currLayer \leftarrow nextLayer
- 14: **until** goFurther
- 15: **return** result

3. FAST 1D 2-MEANS CLUSTERING

There are many existing algorithms for clustering data, and all of those can be applied to the specific problem of clustering 1D data into two clusters. However, the fact that we are only dealing with 1D data and that the number of clusters is fixed to two can be exploited to significantly decrease both the time and the space complexity.

The time complexity of k-means clustering is $O(qknm)$ where q is the number of iterations, k is the number of clusters, n is the number of data points, and m is the dimensionality. Using a general purpose k-means clustering implementation for clustering 1D data should have $O(qkn)$ time complexity. Implementation specifically designed for 1D data can have a different complexity. For example, a method with $O(n^2k)$ is described in [13]. However, this is only an improvement if $n < q$; in most applications, q is much less than n . In the following, we propose a fast 1D 2-means clustering. The proposed algorithm is deterministic and its time complexity is $O(n)$ while maintaining space complexity at $O(1)$.

The 1D k-means algorithm of [13] is slower than the aforementioned general k-means; however, that 1D k-means algorithm is repeatable and optimal, while the general k-means is not. In other words, that 1D k-means algorithm finds the global optimum (in terms of *withinss*). Like this 1D k-means, our proposed 1D 2-means clustering algorithm is repeatable and optimal. But unlike this 1D k-means, it is faster than the general k-means algorithm.

The input data for our algorithm is a list of pairs. Each pair consists of a value (a point in 1D) and its identification (the index of the point, ranging from 1 to n). We develop our algorithm in three steps. Each step yields an improvement over the previous step, either in terms of time or in terms of space complexity.

3.1 Step 1: Sorting-Based Algorithm

In this algorithm, we sort the dataset at first using their values (in ascending order). The best possible computational complexity for sorting is $O(n \log n)$, e.g., with merge sort. The space complexity of merge sort can be as low as $O(1)$ because one can use in-place sorting. Note that the 1D k-means algorithm of [13] assumes that the data is sorted. However, in our application, we need to account for the cost of sorting, since the data needs to be sorted after every projection. Thus, we take into account the time and the space complexity for sorting.

After sorting, we have $x_1 \leq x_2 \leq \dots \leq x_n$ where x_i itself represents the value of pair i ; and i is the identification of x_i . We are interested in the first t elements. The average of the first t elements is the sum of them divided by t . When considering $t+1$ elements, we can do the same thing again. However, if we need to calculate averages for $t = 1, \dots, n$, this is not practical. We can do better using the following formula (see the appendix for derivations):

$$\mu_t = \frac{x_t + (t-1)\mu_{t-1}}{t}. \quad (3.1)$$

Now, we can calculate the *withinss* for the first t values:

$$d^2(x_1, \dots, x_t) = d^2(x_1, \dots, x_{t-1}) + \frac{t-1}{t} (x_t - \mu_{t-1})^2, \quad (3.2)$$

where $d^2(x_1) = 0$ and $t = 2, 3, \dots, n$.

The previous two formulas can be extended to the opposite direction. Let $\mu'_k = \frac{1}{n-k+1} \sum_{j=k}^n x_j$. In other words, μ'_k represents the average from the k_{th} element (inclusive) to the last element (inclusive). By definition, μ'_1 is μ_n or the overall average. For $2 \leq k \leq n-1$,

$$\mu'_k = \frac{-x_{k-1} + (n-k+2)\mu'_{k-1}}{n-k+1}, \quad (3.3)$$

and

$$d^2(x_k, \dots, x_n) = d^2(x_{k-1}, \dots, x_n) - \frac{n-k+2}{n-k+1} (x_{k-1} - \mu'_{k-1})^2. \quad (3.4)$$

First, we calculate the value of μ'_1 and $d^2(x_1, \dots, x_n)$. The time complexity of this pre-computation is $O(n)$. The reason of this pre-computation is to minimize the space

complexity as $O(1)$. Equivalently, if one prefers less computation than memory usage, one can store $d^2(x_1, \dots, x_t)$ for $t = 1, \dots, n$ first. After this first path scanning, one has μ'_1 , which is μ_n , and $d^2(x_1, \dots, x_n)$.

Figure 3.1 and Algorithm 2 present the proposed algorithm with different representations. Note that the number of data points and the standard deviation are fixed for the same dataset. In other words, *normalized withinss* is proportional to *withinss*. Thus, we do not need to have *normalized withinss* for now, as the result of 1D 2-means clustering is the minimum possible *withinss* rather than *normalized withinss*. *Normalized withinss* is calculated from *withinss* later for comparing 1D random projections and finding the best one.

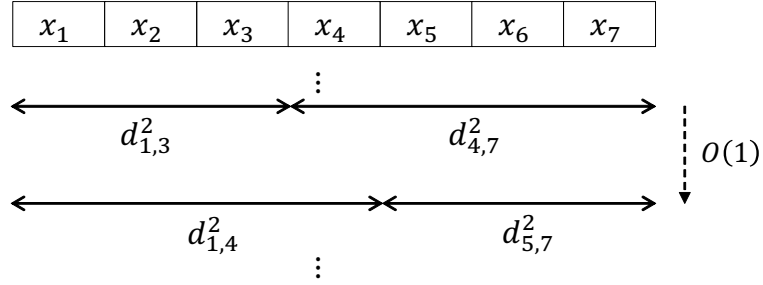


Fig. 3.1.: Complexity of 1D 2-means clustering. We use simpler notation $d^2_{i,j}$ for $d^2(x_i, \dots, x_j)$ because of space limitation. The computational complexity to get $d^2_{1,4}$ from $d^2_{1,3}$ and to get $d^2_{5,7}$ from $d^2_{4,7}$ is $O(1)$. The overall computational complexity is $O(n)$ because there are $n + 1$ steps. The space complexity is $O(1)$ because we do not store the output of each step, but compare to the smallest *withinss* up to that point.

This sorting-based 1D 2-means clustering has $O(n \log n)$ time complexity which comes from the nature of sorting while using a constant amount of memory. If the data is sorted then the complexity becomes linear in terms of the number of data points, $O(n)$.

Algorithm 2 Sorting-based 1D 2-means clustering. s is the threshold to separate the data into two clusters.

Require: $X = \{x_1, \dots, x_n\}$: sorted list

```

1: initialize min_withinss  $\leftarrow MAX$ 
2: for  $t = 1$  to  $n - 1$  do
3:   withinss  $\leftarrow d^2(x_1, \dots, x_t) + d^2(x_{t+1}, \dots, x_n)$ 
4:   if withinss  $<$  min_withinss then
5:     min_withinss  $\leftarrow$  withinss
6:      $s \leftarrow x_t$ 
7:   end if
8: end for
9: return  $s$ , min_withinss

```

3.2 Step 2: Bucket-Based Algorithm

As mentioned earlier, we cannot assume that the data points are sorted in our application. Taking sorting into account, we cannot achieve a lower time complexity than $O(n \log n)$. Thus, to get a lower complexity, we need to bypass sorting entirely.

Our idea, which we call the bucket-based algorithm, is similar in nature to a fast algorithm for computing the median of a dataset without sorting the dataset [16]. However, the details of our proposed algorithm are different.

Figure 3.2 illustrates the bucket-based 1D 2-means clustering. We start with fixed-interval buckets, B_1, \dots, B_q . Determining the bucket size and assigning every data point to the proper bucket have $O(n)$ time complexity. Now, data points are grouped in buckets based on their values. Note that data points in a bucket are not sorted. We know the smallest data point (sp_j) and the largest data point (lp_j) in B_j . Note that $sp_1 = x_1$ and $lp_q = x_n$. Equation 3.2 shows that $d^2(x_1, \dots, x_t)$ is non-decreasing for $t = 1, \dots, n$ while Equation 3.4 shows that $d^2(x_k, \dots, x_n)$ is non-increasing for $k = 1, \dots, n$.

The largest possible *withinss* if partitioning occurs in the bucket is described as the dotted lines in Figure 3.2 (a). That idea can be summarized with the following equation: for $i = 1, \dots, q$,

$$\max(B_i) = d^2(x_1, \dots, lp_i) + d^2(sp_i, \dots, x_n). \quad (3.5)$$

Similarly, the solid lines in Figure 3.2 (a) represent the smallest possible *withinss* in the bucket. They are formalized with the following equation: for $i = 1, \dots, q$,

$$\min(B_i) = d^2(x_1, \dots, sp_i) + d^2(lp_i, \dots, x_n). \quad (3.6)$$

Now we calculate the largest minimum possible value of all q buckets: $\mathbf{lmin} = \min(B_k)$ that satisfies $\min(B_k) \geq \min(B_i)$ for $i = 1, \dots, q$. In Figure 3.2, \mathbf{lmin} is $\min(B_3)$. We can drop B_2 and B_q because the smallest possible *withinss* of B_3 is larger than the largest possible *withinss* of B_2 and B_q . For the remaining buckets, we then do the same thing as described in the sorting-based algorithm.

So our proposed bucket-based algorithm achieves $O(n)$ average time complexity. However, it requires $O(n)$ space complexity because it stores every element.

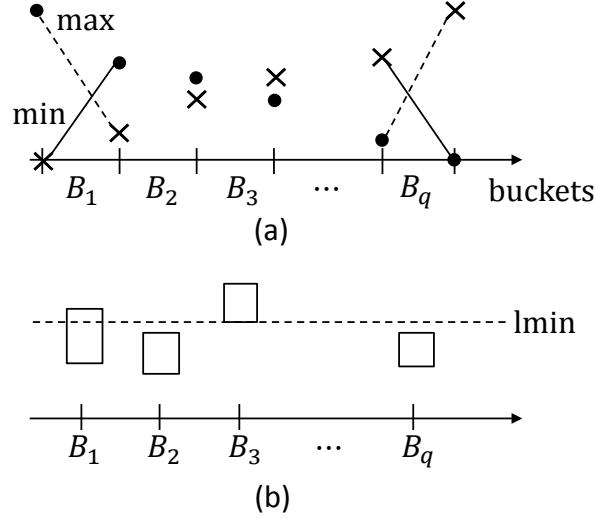


Fig. 3.2.: Illustration of the bucket-based 1D 2-means clustering algorithm. In (a), crosses are $d^2(x_1, \dots, x_t)$ and circles are $d^2(x_k, \dots, x_n)$. For simplicity, we assume that $lp_i = sp_{i+1}$ in this figure.

3.3 Step 3: Partial Quicksort to Construct Buckets

The reason that we can achieve constant space complexity in the sorting-based algorithm is that we use an in-place sorting algorithm. If we can use an in-place algorithm for constructing buckets from data points, then the time complexity goes back to $O(1)$ from $O(n)$.

We use partial quicksort for constructing buckets. As quicksort can be implemented as an in-place algorithm, this can be implemented as an in-place algorithm. However, quicksort in our case finishes at the very initial phase as soon as we have enough buckets.

The average time complexity of quicksort is $O(n \log n)$ because in most cases each iteration split the data well [16]. For the same reason, the average time complexity of partial quick sort is $O(n)$.

After that, the remaining process is exactly the same as the bucket-based algorithm. This has $O(n)$ average time complexity and $O(1)$ space complexity.

4. EXPERIMENTS

We implemented the proposed clustering method using the aforementioned “sorting-based” 1D 2-means algorithm (Section 3.1) using Java. For self-completeness, we did not use any library (e.g., linear algebra libraries) even though this could have yielded a significant performance gain. For short we use the acronym *RP1D* (1D Random Projection) to denote this implementation of our method. The source code is available at <https://engineering.purdue.edu/~mboutin/RP1D>. All computations described in this section were performed on a machine with an Intel i7 2.4GHz CPU and 16 gigabyte of memory.

Our comparison builds on [9], in which ten subspace clustering methods (CLIQUE [17], DOC [18], MINECLUS [19], SCHISM [20], SUBCLU [21], FIRES [22], INSCY [23], PROCLUS [24], P3C [25], and STATPC [26]) were compared using seven datasets (GLASS, VOWEL, PENDIGITS, SHAPE, DIABETES, LIVER, and BREAST). Although the source code was not available, we were able to use the executable software, which was kindly shared publicly on the web¹.

Table 4.1 presents the summary of these seven datasets including the number of data points (size), the number of dimensions (dim), and the number of tag types (classes). Note that all data was acquired from [14]. These seven datasets do not have a very high dimensionality. To further test the scalability of our proposed clustering method, we also tested it on two very high dimensional datasets, as described in Table 4.2. A comparison of the clustering results obtained with our method and with ten other existing methods using these two datasets is described in Section 4.3 after discussing the efficiency (Section 4.1) and the effectiveness (Section 4.2) using the seven datasets of Table 4.1.

¹<http://dme.rwth-aachen.de/en/OpenSubspace>

Table 4.1.: Description of high-dimensional datasets used in experiments.

	GLASS	VOWEL	PENDIGITS	SHAPE	DIABETES	LIVER	BREAST
size	214	990	7494	160	768	345	198
dim	9	10	16	17	8	6	33
classes	6	11	10	9	2	2	2

Table 4.2.: Description of very high-dimensional datasets used in experiments.

	LIBRAS	MFEAT-FAC
size	360	2000
dim	90	216
classes	15	10

Table 4.3.: RP1D parameters.

	From	Offset	Op	Steps	To
threshold	0.01	0.001	+	11	0.02
minSize	0.02	0.005	+	9	0.06
The number of total experiments: 99					

The proposed algorithm (*RP1D*) requires four parameters. The first parameter is the number of repetition (r), which is the number of random projections performed at every node of the hierarchical clustering. We fixed this number to $r = 100$. The second parameter is the histogram bin size, which we set to 0.01. The third parameter is a threshold on `ldiff`, which is used as a termination condition in the hierarchical clustering. The fourth parameter is “minSize”, the minimum allowable size for a cluster, which is also used as a termination condition. In our experiments, the third and fourth parameters were varied as described in Table 4.3. Specifically, the `threshold` for `ldiff` changed from 0.01 to 0.02 as we added 0.001 for each iteration. Thus, there were 11 steps. The `minSize` changed from 0.02 to 0.06 with 9 steps. As a result, the total number of steps was 99.

The parameters for the 10 methods we compared to were varied the same way as in [9]. See the authors’ web page for a detailed description².

4.1 Efficiency

We proposed three algorithms for fast 1D 2-means clustering: a sorting-based algorithm, a bucket-based algorithm, and an algorithm using partial quicksort for constructing buckets. We implemented the sorting-based algorithm and the bucket-based algorithm. We also implemented k-means for comparison. Figure 4.1 shows

²<http://dme.rwth-aachen.de/en/OpenSubspace/evaluation>

that our fast 1D 2-means outperforms k-means (which is faster than 1D k-means [13]). Note that, in comparison with k-means, our fast 1D 2-means is deterministic and finds the global optimum (like 1D k-means [13]).

The data points are generated uniformly randomly in the range of $[0, 1000)$. For each iteration, new data points are generated and clustered using all three algorithms. For the bucket-based algorithm, the number of buckets is fixed at 100 for simplicity. We repeated the experiments 1000 times then average them for each number of data points. Note that k-means clustering is sometimes performed several times and the clustering result with lowest *withinss* is chosen to alleviate the problem of finding local optimum instead of global optimum. However, we did not adopt such a scheme.

In Figure 4.1 (a), we see that k-means is much slower than both our proposed 1D 2-means with sorting and our proposed 1D 2-means with buckets. Indeed, the curves for both of our proposed algorithms are indistinguishable from the horizontal axis. Figure 4.1 (b) shows the run time difference between our two proposed algorithms. We see that even though the bucket-based algorithm is faster as the number of data points increases, the sorting-based algorithm is fast enough (especially when the number of data points is small) while using smaller amount of memory.

The performance of the bucket-based algorithm is mostly affected by the number of buckets that should be sorted. In the experiments, the average number of sorted buckets is 3.5, which is 3.5% of buckets.

We used a sorting-based 1D 2-means for the proposed clustering method because it is easiest of the three to implement and its performance is sufficient as shown in Figure 4.1.

The fast 1D 2-means works as an important component of the proposed clustering algorithm because it is called multiple times for finding the best 1D random projection, deciding the termination condition, and performing 1D binary clustering. In the following, we compare the overall run time of our clustering algorithm with a real dataset.

Table 4.4.: Run-time comparison using PENDIGITS dataset.

	Total	Avg.	Min	Max
PROCLUS	305.65	1.36	0.21	4.72
RP1D	112.40	1.14	0.95	1.56

We used the largest dataset of the seven, PENDIGITS, and the fastest subspace clustering method in [9], PROCLUS. PROCLUS was performed with 224 different parameters in [9] and we followed that parameterization scheme in our experiments.

The run time results are presented in Table 4.4. Observe that although the minimum runtime of PROCLUS is smaller than that of our proposed *RP1D*, its average and maximum are both higher. Thus, *RP1D* tends to be faster, on average, than PROCLUS, and its runtime tends to be more consistent. Note that the runtime we listed for PROCLUS are different from [9] not only because we use a different machine but also because we did not include the time for calculating measures.

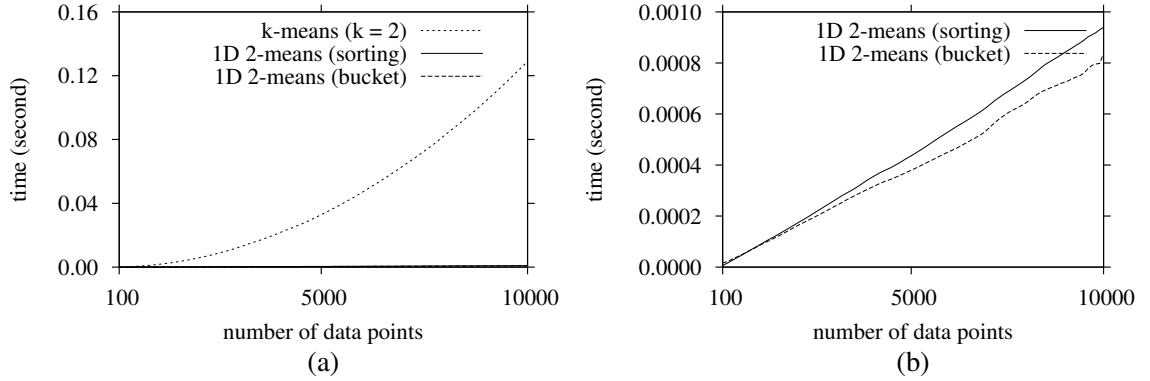


Fig. 4.1.: Runtime comparison between k-means clustering and 1D 2-means clustering as input data size increases.

4.2 Effectiveness

There are several ways to measure the quality of a clustering result. In general, there are internal and external measures. In this experiments, we use external measures for comparison, namely, F1 [23, 27], Accuracy [27, 28], and Entropy [20, 29]. Note that higher values are better in all three measures. Our experimental results are presented in Table 4.5 – 4.11. Note that the numerical values for the 10 subspace clustering methods were copied from [9]. Note also that we measured the *RP1D* clustering results (F1, Accuracy, and Entropy) using the same code as for [9].

F1 is the harmonic mean of precision and recall. Accuracy is the portion of correctly clustered data points. Entropy measures the purity of the clusters found. These three external measures measure the quality of clustering in different ways. However, Entropy can be biased because it is not about finding hidden clusters but only about the purity of the clusters found. In other words, if a subspace clustering method finds only a small number of clear clusters, its Entropy can be very high without being degraded from the fact that many hidden clusters are not revealed. Thus, Entropy should be used only as a subsidiary measure.

Additional to these three external measures, we list the coverage and the number of clusters. As our proposed method is partition based, its coverage is always 1. We are interested in the maximum values of the 11 clustering methods including *RP1D*. The values, which are 5% close to the highest value, are highlighted. The minimum values are used for showing how stable each clustering method is.

Table 4.12 shows the summary of the previous seven tables. For F1, *RP1D* has six highest maximum values among seven. (Note that as the results can be tie, the sum of the columns can be larger than seven). *RP1D* has three best values and six high values in Accuracy. Because of the aforementioned bias in Entropy, we conclude that *RP1D* is generally the most effective clustering method of the 11 clustering methods we compared with.

Table 4.5.: Experimental result using GLASS dataset.

	F1		Accuracy		Entropy		Coverage		Clusters	
	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.51	0.31	0.67	0.50	0.39	0.24	1.00	1.00	6169	175
DOC	0.74	0.50	0.63	0.50	0.72	0.50	0.93	0.91	64	11
MINECLUS	0.76	0.40	0.52	0.50	0.72	0.46	1.00	0.87	64	6
SCHISM	0.46	0.39	0.63	0.47	0.44	0.38	1.00	0.79	158	30
SUBCLU	0.50	0.45	0.65	0.46	0.42	0.39	1.00	1.00	1648	831
FIRES	0.30	0.30	0.49	0.49	0.40	0.40	0.86	0.86	7	7
INSCY	0.57	0.41	0.65	0.47	0.67	0.47	0.86	0.79	72	30
PROCLUS	0.60	0.56	0.60	0.57	0.76	0.68	0.79	0.57	29	26
P3C	0.28	0.23	0.47	0.39	0.43	0.38	0.89	0.81	3	2
STATPC	0.75	0.40	0.49	0.36	0.88	0.36	0.93	0.80	106	27
RP1D	0.79	0.58	0.68	0.51	0.83	0.58	1.00	1.00	84	26

Table 4.6.: Experimental result using VOWEL dataset.

	F1		Accuracy		Entropy		Coverage		Clusters	
	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.23	0.17	0.64	0.37	0.10	0.09	1.00	1.00	3062	267
DOC	0.49	0.49	0.44	0.44	0.58	0.58	0.86	0.86	64	64
MINECLUS	0.48	0.43	0.37	0.37	0.60	0.46	0.98	0.87	64	64
SCHISM	0.37	0.23	0.62	0.52	0.29	0.21	1.00	0.93	494	121
SUBCLU	0.24	0.18	0.58	0.38	0.30	0.13	1.00	1.00	10881	709
FIRES	0.16	0.14	0.13	0.11	0.16	0.13	0.50	0.45	32	24
INSCY	0.82	0.33	0.61	0.15	0.94	0.21	0.90	0.81	163	74
PROCLUS	0.49	0.49	0.44	0.44	0.65	0.65	0.67	0.67	64	64
P3C	0.08	0.05	0.17	0.16	0.13	0.12	0.98	0.95	3	2
STATPC	0.22	0.22	0.56	0.56	0.14	0.14	1.00	1.00	39	39
RP1D	0.61	0.34	0.57	0.30	0.66	0.38	1.00	1.00	86	24

Table 4.7.: Experimental result using PENDIGITS dataset.

	F1		Accuracy		Entropy		Coverage		Clusters	
	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.30	0.17	0.96	0.86	0.41	0.26	1.00	1.00	1890	36
DOC	0.52	0.52	0.54	0.54	0.53	0.53	0.91	0.91	15	15
MINECLUS	0.87	0.87	0.86	0.86	0.82	0.82	1.00	1.00	64	64
SCHISM	0.45	0.26	0.93	0.71	0.50	0.45	1.00	0.93	1092	290
SUBCLU	—	—	—	—	—	—	—	—	—	—
FIRES	0.45	0.45	0.73	0.73	0.31	0.31	0.94	0.94	27	27
INSCY	0.65	0.48	0.78	0.68	0.77	0.69	0.91	0.82	262	106
PROCLUS	0.78	0.73	0.74	0.73	0.90	0.71	0.90	0.74	37	17
P3C	0.74	0.74	0.72	0.72	0.76	0.76	0.90	0.90	31	31
STATPC	0.91	0.32	0.92	0.10	1.00	0.53	0.99	0.84	4109	56
RP1D	0.91	0.79	0.91	0.80	0.88	0.75	1.00	1.00	87	25

Table 4.8.: Experimental result using SHAPE dataset.

	F1		Accuracy		Entropy		Coverage		Clusters	
	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.31	0.31	0.76	0.76	0.66	0.66	1.00	1.00	486	486
DOC	0.90	0.83	0.79	0.54	0.93	0.86	1.00	1.00	53	29
MINECLUS	0.94	0.86	0.79	0.60	0.93	0.82	1.00	1.00	64	32
SCHISM	0.51	0.30	0.74	0.49	0.85	0.55	1.00	0.92	8835	90
SUBCLU	0.36	0.29	0.70	0.64	0.89	0.88	1.00	1.00	3468	3337
FIRES	0.36	0.36	0.51	0.44	0.88	0.82	0.45	0.39	10	5
INSCY	0.84	0.59	0.76	0.48	0.94	0.87	0.88	0.82	185	48
PROCLUS	0.84	0.81	0.72	0.71	0.93	0.91	0.89	0.79	34	34
P3C	0.51	0.51	0.61	0.61	0.80	0.80	0.66	0.66	9	9
STATPC	0.43	0.43	0.74	0.74	0.56	0.56	0.92	0.92	9	9
RP1D	0.96	0.81	0.85	0.38	0.97	0.84	1.00	1.00	79	30

Table 4.9.: Experimental result using DIABETES dataset.

	F1		Accuracy		Entropy		Coverage		Clusters	
	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.70	0.39	0.72	0.69	0.23	0.13	1.00	1.00	349	202
DOC	0.71	0.71	0.72	0.69	0.31	0.24	1.00	0.93	64	17
MINECLUS	0.72	0.66	0.71	0.69	0.29	0.17	0.99	0.96	39	3
SCHISM	0.70	0.62	0.73	0.68	0.34	0.20	1.00	0.79	270	21
SUBCLU	0.74	0.45	0.71	0.68	0.14	0.11	1.00	1.00	1601	325
FIRES	0.52	0.03	0.65	0.64	0.68	0.00	0.81	0.03	17	1
INSCY	0.65	0.39	0.70	0.65	0.44	0.15	0.83	0.73	132	3
PROCLUS	0.67	0.61	0.72	0.71	0.23	0.19	0.92	0.78	9	3
P3C	0.39	0.39	0.66	0.65	0.09	0.07	0.97	0.88	2	1
STATPC	0.73	0.59	0.70	0.65	0.72	0.28	0.97	0.75	363	27
RP1D	0.76	0.67	0.73	0.65	0.37	0.20	1.00	1.00	84	24

Table 4.10.: Experimental result using LIVER dataset.

	F1		Accuracy		Entropy		Coverage		Clusters	
	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.68	0.65	0.67	0.58	0.10	0.02	1.00	1.00	1922	19
DOC	0.67	0.64	0.68	0.58	0.18	0.11	0.99	0.90	45	13
MINECLUS	0.73	0.63	0.65	0.58	0.33	0.16	0.99	0.92	64	32
SCHISM	0.69	0.69	0.68	0.59	0.10	0.08	0.99	0.99	90	68
SUBCLU	0.68	0.68	0.64	0.58	0.07	0.02	1.00	1.00	334	64
FIRES	0.58	0.04	0.58	0.56	0.37	0.00	0.84	0.03	10	1
INSCY	0.66	0.66	0.62	0.61	0.21	0.20	0.85	0.81	166	130
PROCLUS	0.53	0.39	0.63	0.63	0.05	0.05	0.83	0.46	6	2
P3C	0.36	0.35	0.58	0.58	0.02	0.01	0.98	0.94	2	1
STATPC	0.69	0.57	0.65	0.58	0.63	0.05	0.77	0.71	159	4
RP1D	0.76	0.61	0.67	0.52	0.36	0.08	1.00	1.00	88	24

Table 4.11.: Experimental result using BREAST dataset.

	F1		Accuracy		Entropy		Coverage		Clusters	
	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.67	0.67	0.71	0.71	0.26	0.26	1.00	1.00	107	107
DOC	0.73	0.61	0.81	0.76	0.46	0.27	1.00	0.80	60	6
MINECLUS	0.78	0.69	0.78	0.76	0.56	0.37	1.00	1.00	64	32
SCHISM	0.67	0.67	0.75	0.69	0.35	0.34	1.00	0.99	248	197
SUBCLU	0.68	0.51	0.77	0.67	0.27	0.24	1.00	0.82	357	5
FIRES	0.49	0.03	0.76	0.76	1.00	0.01	0.76	0.04	11	1
INSCY	0.74	0.55	0.77	0.76	0.60	0.39	0.97	0.74	2038	167
PROCLUS	0.57	0.52	0.80	0.74	0.32	0.23	0.89	0.69	9	2
P3C	0.63	0.63	0.77	0.77	0.36	0.36	0.85	0.85	28	28
STATPC	0.41	0.41	0.78	0.78	0.29	0.29	0.43	0.43	5	5
RP1D	0.82	0.65	0.80	0.69	0.68	0.34	1.00	1.00	93	26

Table 4.12.: Comparison with existing subspace clustering methods.

	F1		Accuracy		Entropy	
	best	5%	best	5%	best	5%
CLIQUE	0	0	2	5	0	0
DOC	0	1	2	3	0	1
MINECLUS	0	4	0	3	0	1
SCHISM	0	0	2	4	0	0
SUBCLU	0	1	0	3	0	0
FIRES	0	0	0	0	1	1
INSCY	1	1	0	4	1	2
PROCLUS	0	0	0	2	0	1
P3C	0	0	0	1	0	0
STATPC	1	3	0	4	4	4
RP1D	6	6	3	6	1	1

4.3 Very High-Dimensional Data

One advantage of our proposed method is that it scales up well to very high dimensional datasets. To illustrate this, we used two more datasets with very high dimensionality: LIBRAS (Table 4.13) and MFEAT-FAC (Table 4.14). In the experiments with LIBRAS ($m = 90$), we had to exclude the results of MINECLUS, FIRES, and STATPC from our experiments because they yielded exceptional or nonsensical results. We also excluded CLIQUE, SCHISM, SUBCLU, and INSCY because they did not finish after running for 12 hours. Table 4.13 shows the comparison between *RP1D* and the three remaining methods: DOC, PROCLUS, and P3C.

In all three quality measures (F1, Accuracy, and Entropy), *RP1D* has the highest maximum and minimum values. In addition, it is the most efficient in terms of average and maximum runtimes.

As we increased the dimensionality to 216 with MFEAT-FAC, DOC and P3C did not finish within 12 hours. As a result the only comparable clustering method is PROCLUS. Now, *RP1D*'s maximum runtime is even smaller than the minimum runtime of PROCLUS. This indicates that *RP1D* is more efficient than PROCLUS in very high dimensions while maintaining superior effectiveness.

Table 4.13.: Comparison using LIBRAS dataset.

	F1		Accuracy		Entropy		Coverage		NumClusters		Runtime		
	max	min	max	min	max	min	max	min	max	min	max	min	avg
DOC	0.11	0.01	0.18	0.06	0.28	0.00	1.00	0.87	3	1	514.34	2.94	132.42
PROCLUS	0.59	0.02	0.46	0.08	0.79	0.03	0.99	0.39	54	2	1.27	0.10	0.47
P3C	0.35	0.00	0.28	0.06	0.83	0.62	0.58	0.00	29	0	6.38	2.89	1.26
RP1D	0.83	0.51	0.73	0.42	0.87	0.62	1.00	1.00	86	25	0.47	0.11	0.16

Table 4.14.: Comparison using MFEAT-FAC dataset.

	F1		Accuracy		Entropy		Coverage		NumClusters		Runtime		
	max	min	max	min	max	min	max	min	max	min	max	min	avg
PROCLUS	0.53	0.05	0.51	0.14	0.63	0.02	1.00	0.51	54	2	21.11	3.17	7.63
RPID	0.84	0.72	0.84	0.71	0.81	0.65	1.00	1.00	81	23	2.45	1.41	1.69

5. CONCLUSION

Conventional clustering methods like k-means [30] and DBSCAN [6] are ill-suited for dealing with high-dimensional data. This is because of the well known curse of dimensionality. There has been a lot of effort in trying to develop effective and efficient methods for clustering high-dimensional data. For example, CLIQUE [17], SCHISM [20], SUBCLU [21], and INSCY [23] can successfully handle about 40 dimensions. However, our experiments indicated that they do not scale up well to 90 dimensions or more.

To address this issue, we propose a clustering method based on a hierarchy of binary clusterings. Each binary clustering is obtained by random projection onto a 1D space. Indeed, real data often has a lot of structure, so much so that a 1D random projection of the data often has a high probability to have two well separated clusters, as we showed experimentally. In other words, each 1D random projection can capture part of that structure (grouping) with a high probability. To increase the chance of successfully capturing a meaningful grouping, we repeat the projection (about 100 times) and keep the “best” one. To measure the quality of the grouping, we use a quantity called *normalized withinss*, which is obtained by dividing *withinss* [13] by the variance of the projected points and the number of data points. We then perform 1D binary clustering of the projected values.

Clustering 1D data is a simple task that can be done with a variety of methods. To further increase the efficiency of our method, we developed a fast 1D 2-means clustering method. Our fast 1D 2-means clustering has $O(n)$ average time complexity and $O(1)$ space complexity while still being guaranteed to find the global optimum.

Our clustering method allows us to effectively cluster very high-dimensional data (several hundreds) in a very small amount of time. For example, when clustering a 216 dimensional dataset (MFEAT-FAC), our method returns a result in under 3 seconds.

This is more than 4.5 times faster than the only other algorithm we found that could handle such a high dimensionality (PROCLUS [24]) within 12 hours.

While PROCLUS seems to be able to handle data with more than 200 dimensions, its accuracy is hardly comparable to that of other existing methods, such as STATPC [26]. In contrast, our method yields very accurate results both for medium dimensionality (6D – 33D) and for very high dimensionality (216D). For example, in 6 out of 7 comparisons we performed, it gave the best F1 [23, 27] measure among all methods compared. It also scored top in 3 out of 7 comparisons in terms of Accuracy [27, 28]. No single method had a comparable number of best scores. When it comes to very high-dimensional data (90 and 216 dimensions), the comparison becomes even more clear in all three external measures (F1, Accuracy, and Entropy [20, 29]).

There are many ways to potentially improve our methods. For example, different random models could be used to generate the 1D vector onto which the data is projected. For simplicity, we chose to generate each entry of the vector independently following a uniform distribution. However, other random models might yield better result. One could also use another measure of separation for the 1D projected data. We chose to use *normalized withinss* for computational reasons, but there may be other, more accurate measures. Our source code is fully accessible at <https://engineering.purdue.edu/~mboutin/RP1D> and other should feel free to modify it to suit their particular applications or try to improve on it.

REFERENCES

REFERENCES

- [1] J. A. Hartigan and M. A. Wong, “Algorithm as 136: A k-means clustering algorithm,” *Applied statistics*, pp. 100–108, 1979.
- [2] A. K. Jain, “Data clustering: 50 years beyond k-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.
- [3] B. Schölkopf, A. Smola, and K.-R. Müller, “Nonlinear component analysis as a kernel eigenvalue problem,” *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.
- [4] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the em algorithm,” *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38, 1977.
- [5] T. Zhang, R. Ramakrishnan, and M. Livny, “Birch: an efficient data clustering method for very large databases,” in *ACM SIGMOD Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.
- [6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in *Kdd*, vol. 96, 1996, pp. 226–231.
- [7] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is “nearest neighbor” meaningful?” in *Database Theory - ICDT’99*. Springer, 1999, pp. 217–235.
- [8] M. J. Zaki and W. Meira Jr, *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, 2014.
- [9] E. Müller, S. Günnemann, I. Assent, and T. Seidl, “Evaluating clustering in subspace projections of high dimensional data,” *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1270–1281, 2009.
- [10] S. Dasgupta, “Learning mixtures of gaussians,” in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 1999, pp. 634–644.
- [11] —, “Experiments with random projection,” in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2000, pp. 143–151.
- [12] W. B. Johnson and J. Lindenstrauss, “Extensions of lipschitz mappings into a hilbert space,” *Contemporary mathematics*, vol. 26, no. 189-206, p. 1, 1984.
- [13] H. Wang and M. Song, “Ckmeans. 1d. dp: Optimal k-means clustering in one dimension by dynamic programming.” *R Journal*, vol. 3, no. 2, 2011.

- [14] S. Hettich and S. Bay, “The uci kdd archive [<http://kdd.ics.uci.edu>]. irvine, ca: University of california,” *Department of Information and Computer Science*, 1999.
- [15] Wharton research data services. Online. Center for Research in Security Prices (CRSP) at the University of Chicago. [Online]. Available: <http://wrds-web.wharton.upenn.edu/wrds/>
- [16] S. S. Skiena, *The Algorithm Design Manual*. Springer, London, 2008.
- [17] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications*. ACM, 1998, vol. 27, no. 2.
- [18] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. Murali, “A monte carlo algorithm for fast projective clustering,” in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 2002, pp. 418–427.
- [19] M. L. Yiu and N. Mamoulis, “Frequent-pattern based iterative projected clustering,” in *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE, 2003, pp. 689–692.
- [20] K. Sequeira and M. Zaki, “Schism: A new approach for interesting subspace mining,” in *Data Mining, 2004. ICDM’04. Fourth IEEE International Conference on*. IEEE, 2004, pp. 186–193.
- [21] K. Kailing, H.-P. Kriegel, and P. Kröger, “Density-connected subspace clustering for high-dimensional data,” in *Proc. SDM*, vol. 4. SIAM, 2004.
- [22] H.-P. Kriegel, P. Kroger, M. Renz, and S. Wurst, “A generic framework for efficient subspace clustering of high-dimensional data,” in *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005, pp. 8–pp.
- [23] I. Assent, R. Krieger, E. Muller, and T. Seidl, “Inscy: Indexing subspace clusters with in-process-removal of redundancy,” in *Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 719–724.
- [24] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, “Fast algorithms for projected clustering,” in *ACM SIGMOD Record*, vol. 28, no. 2. ACM, 1999, pp. 61–72.
- [25] G. Moise, J. Sander, and M. Ester, “P3c: A robust projected clustering algorithm,” in *Data Mining, 2006. ICDM’06. Sixth International Conference on*. IEEE, 2006, pp. 414–425.
- [26] G. Moise and J. Sander, “Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering,” in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 533–541.
- [27] E. Müller, I. Assent, R. Krieger, S. Günnemann, and T. Seidl, “Densest: Density estimation for data mining in high dimensional spaces.” in *SDM*. SIAM, 2009, pp. 175–186.

- [28] B. Bringmann and A. Zimmermann, “The chosen few: On identifying valuable patterns,” in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE, 2007, pp. 63–72.
- [29] I. Assent, R. Krieger, E. Muller, and T. Seidl, “Dusc: Dimensionality unbiased subspace clustering,” in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE, 2007, pp. 409–414.
- [30] J. MacQueen *et al.*, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. California, USA, 1967, pp. 281–297.

APPENDIX

A. DERIVATIONS

$$\begin{aligned}
 \mu_t &= \frac{1}{t} \sum_{j=1}^t x_j, \\
 &= \frac{1}{t} \left(\frac{t-1}{t-1} \sum_{j=1}^{t-1} x_j + x_t \right), \\
 &= \frac{x_t + (t-1) \mu_{t-1}}{t}.
 \end{aligned}$$

$$\begin{aligned}
 d^2(x_1, \dots, x_t) &= \sum_{j=1}^t (x_j - \mu_t)^2, \\
 &= \sum_{j=1}^t \left(x_j - \frac{x_t + (t-1) \mu_{t-1}}{t} \right)^2, \\
 &= \sum_{j=1}^t \left(x_j - \mu_{t-1} - \frac{x_t - \mu_{t-1}}{t} \right)^2, \\
 &= \sum_{j=1}^t (x_j - \mu_{t-1})^2 \\
 &\quad - 2 \cdot \frac{x_t - \mu_{t-1}}{t} \cdot \sum_{j=1}^t (x_j - \mu_{t-1}) + \frac{(x_t - \mu_{t-1})^2}{t}, \\
 &= d^2(x_1, \dots, x_{t-1}) + (x_t - \mu_{t-1})^2 \\
 &\quad - 2 \cdot \frac{x_t - \mu_{t-1}}{t} \cdot (x_t - \mu_{t-1}) + \frac{(x_t - \mu_{t-1})^2}{t}, \\
 &= d^2(x_1, \dots, x_{t-1}) + (x_t - \mu_{t-1})^2 \left(1 - \frac{2}{t} + \frac{1}{t} \right), \\
 &= d^2(x_1, \dots, x_{t-1}) + \frac{t-1}{t} (x_t - \mu_{t-1})^2.
 \end{aligned}$$

$$\begin{aligned}
\mu'_k &= \frac{1}{n-k+1} \sum_{j=k}^n x_j, \\
&= \frac{1}{n-k+1} \left(\frac{n-k+2}{n-k+2} \sum_{j=k-1}^n x_j - x_{k-1} \right), \\
&= \frac{-x_{k-1} + (n-k+2) \mu'_{k-1}}{n-k+1}.
\end{aligned}$$

$$\begin{aligned}
d^2(x_k, \dots, x_n) &= \sum_{j=k}^n (x_j - \mu'_k)^2, \\
&= \sum_{j=k}^n \left(x_j - \frac{-x_{k-1} + (n-k+2) \mu'_{k-1}}{n-k+1} \right)^2, \\
&= \sum_{j=k}^n \left(x_j - \mu'_{k-1} - \frac{\mu'_{k-1} - x_{k-1}}{n-k+1} \right)^2, \\
&= \sum_{j=k}^n (x_j - \mu'_{k-1})^2 \\
&\quad - 2 \cdot \frac{\mu'_{k-1} - x_{k-1}}{n-k+1} \sum_{j=k}^n (x_j - \mu'_{k-1}) \\
&\quad + \frac{(\mu'_{k-1} - x_{k-1})^2}{n-k+1}, \\
&= d^2(x_{k-1}, \dots, x_n) - (x_{k-1} - \mu'_{k-1})^2 \\
&\quad - \frac{2(x_{k-1} - \mu'_{k-1})^2}{n-k+1} + \frac{(x_{k-1} - \mu'_{k-1})^2}{n-k+1}, \\
&= d^2(x_{k-1}, \dots, x_n) \\
&\quad - \frac{n-k+1+2-1}{n-k+1} (x_{k-1} - \mu'_{k-1})^2, \\
&= d^2(x_{k-1}, \dots, x_n) - \frac{n-k+2}{n-k+1} (x_{k-1} - \mu'_{k-1})^2.
\end{aligned}$$

VITA

VITA

Sangchun Han received a B.S. degree from the School of Information and Communication Engineering, Sungkyunkwan University, Korea in February 2007. He received an M.S. degree from the Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Korea in February 2009. In August 2009, he began pursuing a Ph.D. degree from the School of Electrical and Computer Engineering, Purdue University, West Lafayette, USA.